Lively Wiki
A Development Environment for Creating and Sharing Active Web Content

Robert Krahn
Hasso-Plattner-Institut,
University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
Potsdam, Germany
robert.krahn@hpi.uni-potsdam.de

Dan Ingalls
Sun Microsystems Laboratories
16 Network Circle
Menlo Park
dan.ingalls@sun.com

Robert Hirschfeld
Hasso-Plattner-Institut,
University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
Potsdam, Germany
robert.hirschfeld@hpi.uni-potsdam.de

Jens Lincke
Hasso-Plattner-Institut,
University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
Potsdam, Germany
jens.lincke@hpi.uni-potsdam.de

Krzysztof Palacz
Sun Microsystems Laboratories
16 Network Circle
Menlo Park
krzysztof.palacz@sun.com

ABSTRACT

Wikis are Web-based collaborative systems designed to help people share information. Wikis have become popular due to their openness which gives users complete control over the organization and the content of wiki pages. Unfortunately existing wiki engines restrict users to enter only passive content, such as text, graphics, and videos and do not allow users to customize wiki pages. Thus, wikis cannot be used to host or author rich dynamic and interactive content.
In this paper we present Lively Wiki, a development and collaboration environment based on the Lively Kernel which enables users to create rich and interactive Web pages and applications – without leaving the Web. Lively Wiki combines the wiki metaphor with a direct-manipulation user interface and adds a concept for Web programming as well as programming tool support to create an easy to use, scalable, and extendable Web authoring tool. Moreover, Lively Wiki is self-supporting, i.e. the development tools were used for creating its own implementation thereby giving users the freedom to customize every aspect of the system.

Categories and Subject Descriptors

D.2.6 [**Programming Environments**]; D.2.2 [**Design Tools and Techniques**]; D.3 [**Programming Languages**]; H.5.4 [**Hypertext/Hypermedia**]

General Terms

Design, Human Factors

Keywords

Wikis, Application Wikis, Web Application, Morphic, User Innovation, Development Environment, End-user Programming

1. INTRODUCTION

During the last decade the Internet and especially the World Wide Web have become more and more a platform for applications which are replacing traditional desktop software. This paradigm shift towards Web-based software seems to continue [36, 34] and there are good reasons. Software in the Web is platform independent, it can be accessed from all over the world, upgrades can be done immediately and centrally without affecting users, usually no installation is necessary to run it, and users can interact and collaborate. However, using the Web as an application platform also has several drawbacks. The static publishing model of server-hosted hypertext and the strict client/server architecture make creating applications for the Web different from and often more complicated than creating applications for desktop environments [36].

In detail this means that developers have to integrate several different technologies at once [12, 39]: (X)HTML, XML, Cascading Style Sheets, JavaScript and the Document Object Model (DOM) interface, PHP, ASP, Java or another programming language for server side programming. Additionally there is the inherent multi-tier architecture which forces a conceptual separation into a presentation layer and domain/backup layer (server), resulting in complex software architectures using the Model-View-Controller [11] or Model-View-Presenter [10] design pattern even for simple applications.

These technical obstacles make it hard to create interactive, rich, and lightweight applications. In comparison, non Web-based development and authoring environments like HyperCard [3, 15] and Smalltalk programming environments [13], especially Squeak [19], make it easy to create these kind of applications due to their graphical and flexible UIs, uncomplicated programming interfaces, and direct development models.

With Lively Wiki we want to combine the advantages of the Web with the development capabilities of the mentioned authoring systems to create a general purpose Web programming environment that allows the creation of interactive and dynamic applications. To allow openness, incremental changes, and collaboration we combined wiki principles [8] with a Web development model and programming tools. To implement our system we extended the Lively Kernel, a Web framework providing a convenient abstraction of different Web technologies [20].

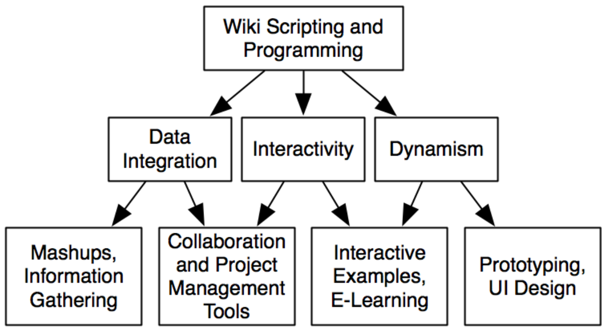

Wiki Scripting and Programming
Data Integration
Interactivity
Dynamism
Mashups, Information Gathering
Collaboration and Project Management Tools
Interactive Examples, E-Learning
Prototyping, UI Design

Figure 1: Active wikis enable new usage scenarios

Additionally to reported use cases for End-user Web development [31, 4] we can imagine several other examples for using programming capabilities in the Web. Figure 1 shows a summary of use case categories. Content whose main purpose it is to convey knowledge can be made more informative and more vivid by combining interactive and multimedia elements. Wikipedia articles could have user created interactive examples attached making certain topics more comprehensible. Articles would not just have a static form but combine text with non-linear complex systems into Active Essays as described by Kay [21, 44]. Project teams could rapidly design and implement their own tools for supporting their workflow, for example a collaborative calendar as presented in section 5. Applications could gather information from different sources and create information compendiums and mashups. An accessible and interactive environment allows also to quickly create prototypes and try out UI designs. Of course those applications could be individually combined.

The remainder of the paper is organized as follows. Section 2 gives a short overview of the state of the art of programming in the Web. Section 3 describes the concepts and ideas behind Lively Wiki. Section 4 discusses the implementation of our system. Section 5 presents applications that were developed in the wiki. Section 6 discusses the related work and section 7 summarizes the work and gives an outlook.

2. CLIENT SIDE AUTHORING AND PROGRAMMING IN THE WEB

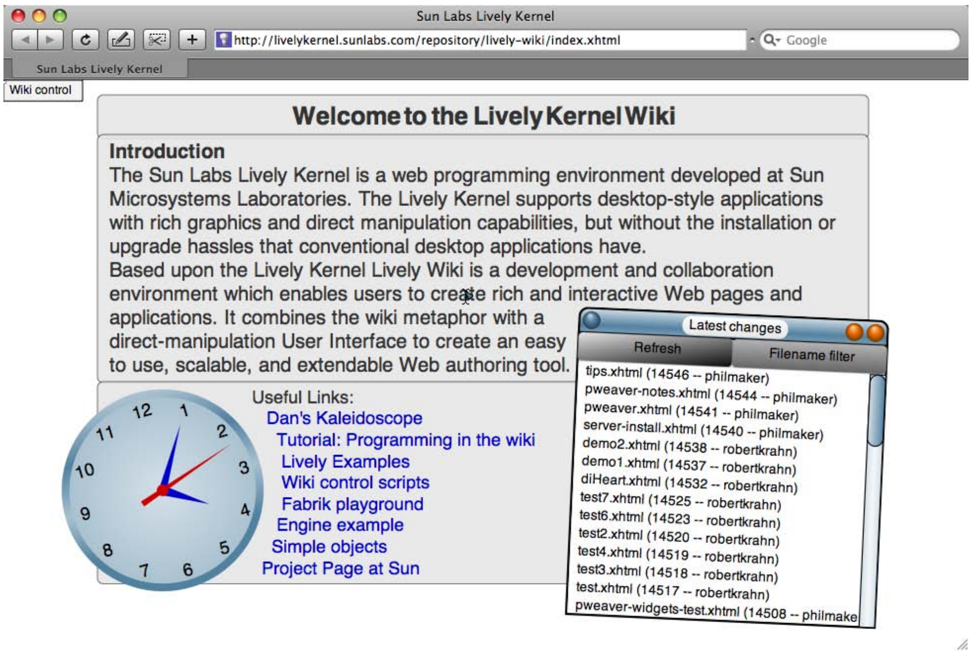
Web programming is usually done on the Web server side using technologies like the Common Gateway Interface (CGI), Web server extensions with modules, or Web frameworks that integrate the different Web technologies [39]. To create or change these applications, server access and knowledge about the different Web technologies is required. Although Web browsers support dynamic scripting since the introduction of JavaScript these capabilities were seldom used to enable users to influence the behavior of Web programs. Client side Web development is emerging as a new field of software engineering empowering users to create applications by using nothing else than a Web browser [35]. The purpose of these programming systems is mainly to create situational applications [33] which allow users to automate tasks or integrate data. We found several different types of implementations:

- Bookmarklets are bookmarks that execute a JavaScript expression [30]
- Programming systems implemented as Web browser extensions [6] allow to define and execute actions on DOM elements
- Mashups dynamically combine content from more than one Web site [34]
- Application Wikis combine wiki markup with scripting languages
- Active Web Essays [44] bring the authoring of Active Essays [21] to the Web

These tools bring development capabilities to end-users, however, they are mostly domain specific and do not allow general purpose programming. It was argued that end-users need robust and constrained environments in order to create productive results [4]. We agree with that statement, however, we do not believe that a platform used to create end-user programming tools should be constrained per se. We therefore base our work on the Lively Kernel.

The Lively Kernel is a client side platform for Web application development running in modern Web browsers without installation and is implemented entirely in JavaScript [20, 35]. The goal of the Lively Kernel project is to create a framework in which application development should not be ”more complicated, less general, or any less fun than other modes of programming” [20]. The Lively Kernel implements the composable graphics system Morphic [26] which can be used to display and animate objects and process user inputs. The current version uses SVG and canvas as underlaying graphic libraries, however, it provides a complete abstraction from the interfaces of those systems and users interact with the Morphic interface. Lively Kernel's interface also has support for networking (abstracting XMLHttpRequest), provides an object system, and has several other programming conveniences.

Although the Lively Kernel provides tools for browsing and editing source code those tools proved to be tedious to handle, were not able to show the source code for all parts of the system, and, most important, were not sufficient for general development purposes, especially not for programming in a wiki.


Sun Labs Lively Kernel

http://livelykernel.sunlabs.com/repository/lively-wiki/index.xhtml

Wiki control

Welcome to the Lively Kernel Wiki

Introduction

The Sun Labs Lively Kernel is a web programming environment developed at Sun Microsystems Laboratories. The Lively Kernel supports desktop-style applications with rich graphics and direct manipulation capabilities, but without the installation or upgrade hassles that conventional desktop applications have.
Based upon the Lively Kernel Lively Wiki is a development and collaboration environment which enables users to create rich and interactive Web pages and applications. It combines the wiki metaphor with a direct-manipulation User Interface to create an easy to use, scalable, and extendable Web authoring tool.

Useful Links:
Dan's Kaleidoscope
Tutorial: Programming in the wiki
Lively Examples
Wiki control scripts
Fabrik playground
Engine example
Simple objects
Project Page at Sun

Latest changes

Refresh | Filename filter

tips.xhtml (14546 -- philmaker)
pweaver-notes.xhtml (14544 -- philmaker)
pweaver.xhtml (14541 -- philmaker)
server-install.xhtml (14540 -- philmaker)
demo2.xhtml (14538 -- robertkrahn)
demo1.xhtml (14537 -- robertkrahn)
diHeart.xhtml (14532 -- robertkrahn)
test7.xhtml (14525 -- robertkrahn)
test6.xhtml (14523 -- robertkrahn)
test2.xhtml (14520 -- robertkrahn)
test4.xhtml (14519 -- robertkrahn)
test3.xhtml (14518 -- robertkrahn)
test.xhtml (14517 -- robertkrahn)
pweaver-widgets-test.xhtml (14508 -- philmake

Figure 2: Lively Wiki start world

3. LIVELY WIKI: CREATING AND SHARING ACTIVE CONTENT

Lively Wiki is a development and collaboration environment based on the Lively Kernel that enables users to create rich and interactive Web pages and applications. It combines the wiki metaphor with a direct-manipulation user interface to create an easy to use, scalable, and extendable Web authoring tool.

3.1 Wiki Structure and Principles

The primary unit for structuring content in Lively Wiki is a world. Worlds are web pages that are identified by a URL and can be loaded with a Web browser. Doing this, the world content is loaded and the Lively Wiki system is started. A user can than interact with and change the contents of that world. Worlds can include links to other worlds. Creating new worlds can be done by clicking on a link to a not existing world or by saving an existing world with a different name. In both cases the new world is a clone of the old one.

Lively Wiki follows the wiki design principles [8]. Lively Wiki is a open system, allowing users to change worlds as they see fit[1]. It allows incremental changes by not requiring referenced worlds to be existing yet and by allowing users to define world content and behavior gradually. This enables Lively Wiki to support growth and content evolution, it hence has an organic structure[2]. The universal principle applies because the mechanisms of organizing the world structure and editing text content are the same, as are the mechanisms for changing page behavior and the overall wiki behavior. Furthermore, Lively Wiki is tolerant to errors as non-affected behavior will continue to work when errors in other parts exist. When errors occur in basic system parts Lively Wiki allows users to revert a world to an older working version.

[1]The current prototype allows write access only for registered users, however, requiring an user account for modifying the content can be made optional.

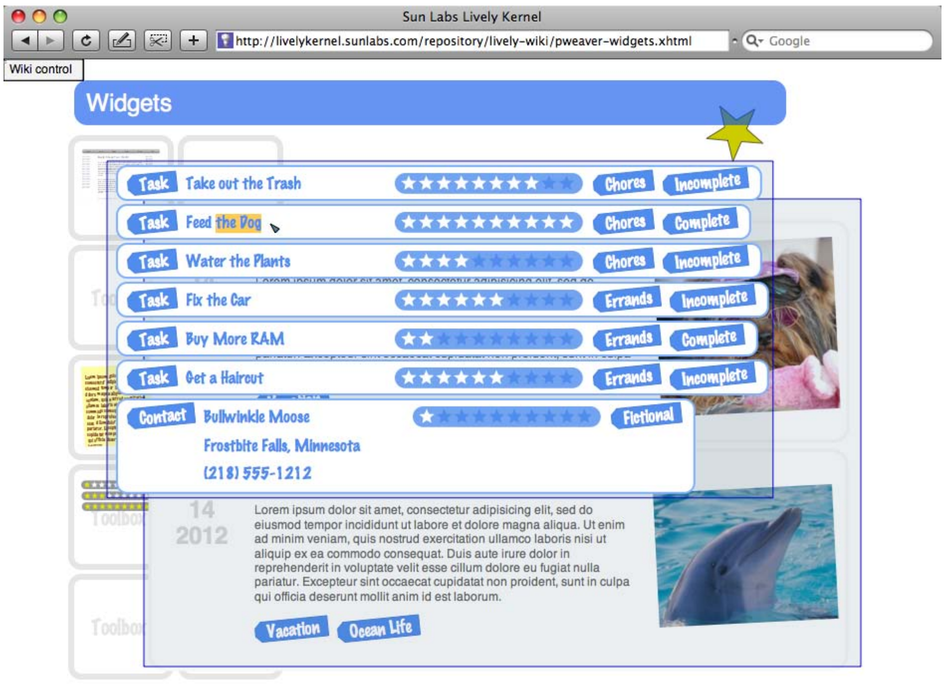
[2]See section 5 for a visualization of the world structure and its changes.

3.2 User interface

Lively Wiki's goal is to make the user-system interaction easy and straightforward. Instead of having editing modes and markup languages for modifying wiki worlds we prefer a direct, more desktop-like user interaction. We use Lively Kernel's Morphic implementation. Morphic is a user interface construction environment [27, 26]. Its two main principles are directness and liveness. Directness means that there is no intermediate representation or mode when manipulating objects. Liveness means that screen objects are interactive and active, i.e. they react to user input and can have a custom behavior. These principles allow the iterative creation of graphical interfaces in direct manner. This means that screen objects (Morphs) can be manipulated directly with a pointing device. To edit text, for example, a user clicks into a text morph and starts editing. Via the context menu or with shortcuts text attributes can be changed. Of course editing capabilities for morphs can also be turned off individually.

In the same way other morphs can be modified and created. Users are able to layout and compose complex worlds by clicking and dragging objects. No predefined style limits world layouts (although such layout policies can be individually defined, however, currently Lively Wiki has only simple flow layouts). There exist several predefined morph types, the most important are: Rectangle, ellipse, text, image, and video morphs.

Figure 3 shows an example for a user defined world and applications: The Todo-application in the foreground was composed out of other morphs. Users can create new entries by copying old entries and changing their labels.


Sun Labs Lively Kernel
http://livelykernel.sunlabs.com/repository/lively-wiki/pweaver-widgets.xhtml
Google
Wiki control
Widgets
Task Take out the Trash Chores Incomplete
Task Feed the Dog Chores Complete
Task Water the Plants Chores Incomplete
Task Fix the Car Errands Incomplete
Task Buy More RAM Errands Complete
Task Get a Haircut Errands Incomplete
Contact Bullwinkle Moose Fictional
Frostbite Falls, Minnesota
(218) 555-1212
14
2012
Vacation
Ocean Life
Toolbox

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figure 3: A Lively Wiki world showing user created applications[1]

3.3 Programming with Lively Wiki

When creating the programming capabilities for Lively Wiki we had two goals: To not restrict the expressiveness and the programming potential and at the same time make programing in the wiki not harder than editing other content of wiki pages. The Lively Kernel already had tools for browsing and editing source code built into it. However those tools proved to be tedious to handle, were not able to show the source code for all parts of the system, and, most important, were not sufficient for general development purposes, especially not for programming in a wiki. In a wiki, changes should not affect the entire system but should be applied to local worlds only. Therefore, we created a development model and tools which specifically support wiki programming.

3.3.1 Programming Model and Process

In addition to editing the static content of worlds, Lively Wiki users can also define the behavior of objects in those worlds. We do not restrict the programmability of the wiki to certain use cases like creating mashups; users can change the system in a general and not predefined manner. Because the wiki is completely implemented in itself there are no restrictions and literally every aspect of the system can be changed.

When a world without changes is loaded, the behavior of objects is defined by the base system of Lively Wiki. Changes can then be done incrementally to the base system but apply only locally. Instead of saving those changes to the Lively Wiki source code files, they are attached to a ChangeSet of the world. The ChangeSet is reinvoked every time the world is loaded. Figure 4 shows a model of three worlds. Every world has a ChangeSet attached. Cloned

[1]The author is Philip Weaver

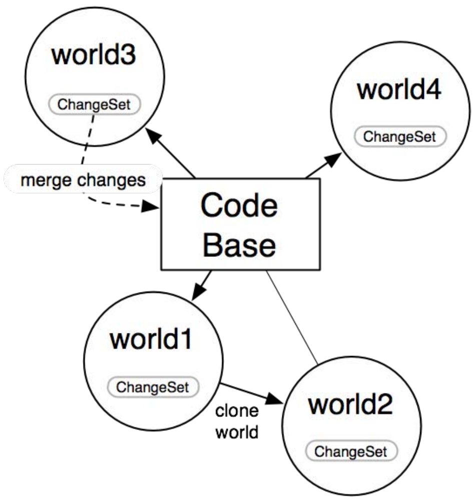


Figure 4: Software development scheme in a wiki

worlds like world2 inherit the changes from the world they are created from. Wiki users can only manipulate worlds. Wiki developers, however, can also change the base system. Accordingly, they can also manually merge changes from worlds back into the base system.

The world model allows users to make changes iteratively and publish them instantly. It is possible to collaborate asynchronously when programming in the same world (section 4.2 for details). A wiki world in this sense is an environment that has all the necessary development tools and is, at the same time, the platform for running the applications. This has a lot of advantages. Changes can be immediate, there are no compile-run cycles. This helps to discover and correct errors and makes the development process faster and more dynamic. Moreover, there is no deployment process, because the application is always running in the world where it is developed. Of course, worlds can be cloned to make "releases" of applications. The tooling required for development is already built into the wiki, so users do not need to install and learn additional tools.

In a programming environment where everything can be modified the danger exists that users break functionality. However, this is not a serious problem because changes apply only to a local world and can be reverted.

3.3.2 *Programming Tools*

Most of the programming tasks like defining a new class could be done from a simple text morph (source code can be evaluated from everywhere) but this is cumbersome and changes would also not persist. Therefore Lively Wiki's development environment extends the Lively Kernel tooling with a system browser for viewing and manipulating all source code, a debugger-like stack viewer (using Lively Kernels shadow stack feature [20]), and a test framework.

System Browser.

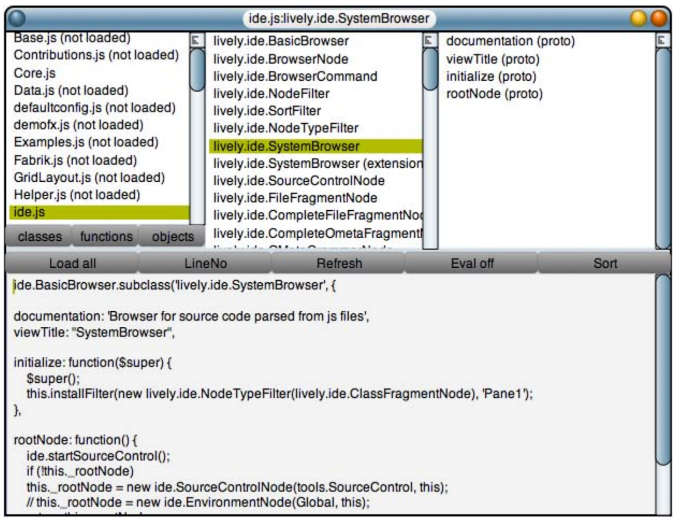
The main development tool for wiki programmers is the SystemBrowser. The browser parses the source code of Lively Wiki, extracts semantic information, and then displays it in the browser. This allows users to deal directly with classes, methods, and functions, thereby making code reading and writing much easier as well as forcing programmers to write clean and structured code [14]. The three-structured display allows navigating and visualizing the structure of code similar to the original Smalltalk-80 class browser [13]. However, the SystemBrowser is adapted to the requirements of Web programming with JavaScript. Code changes are evaluated at runtime and affect the world behavior instantly.

The browser is written in a general way based on the OmniBrowser architecture [5] and thus can be extended to display not only the source code of files but also any other tree structured information. Currently the browser is able to display source code of plain JavaScript files, Lively Kernel's lkml definitions, and OMeta grammars [41].

The upper half of the SystemBrowser as shown in Figure 5 is divided in three panes, the left one showing class categories (these are files and the ChangeSet), the middle pane displays classes, functions, or objects, and the right pane shows methods or functions. The buttons in the middle allow users to enable/disable context independent functionality. For example the file line numbers can be displayed or the list items sorted. Context-dependent functionality can be found in context menus. When invoking the context menu of a method, for example, options to browse the implementors and senders of that method, to remove or to clone the method are displayed. The browser also supports a lot of programming conveniences like bracket matching, in-line evaluation, and allows to search and browse the code base.

Stack Viewer.

The Stack viewer (see figure 6) can be invoked when placing a `halt()` call in the source code. When the control-flow reaches the spot the viewer showing the current call stack is opened. The source code of methods or functions can then be browsed. In-line evaluation is enabled so users can



```
Base.js (not loaded)
Contributions.js (not loaded)
Core.js
Data.js (not loaded)
defaultconfig.js (not loaded)
demofx.js (not loaded)
Examples.js (not loaded)
Fabrik.js (not loaded)
GridLayout.js (not loaded)
Helper.js (not loaded)
ide.js
```

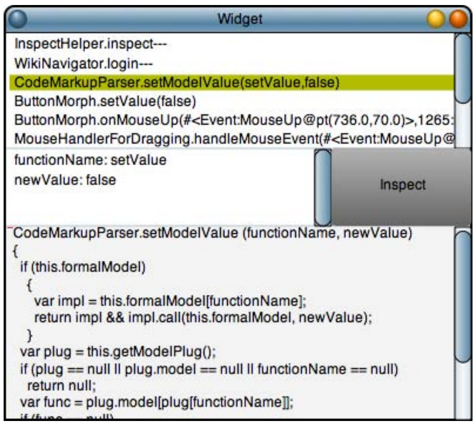
```
lively.ide.BasicBrowser
lively.ide.BrowserNode
lively.ide.BrowserCommand
lively.ide.NodeFilter
lively.ide.SortFilter
lively.ide.NodeTypeFilter
lively.ide.SystemBrowser
lively.ide.SystemBrowser (extension
lively.ide.SourceControlNode
lively.ide.FileFragmentNode
lively.ide.CompleteFileFragmentNo
lively.ide.CompleteOmetaFragment
```

```
documentation (proto)
viewTitle (proto)
initialize (proto)
rootNode (proto)
```

```
ide.BasicBrowser.subclass('lively.ide.SystemBrowser', {

documentation: 'Browser for source code parsed from js files',
viewTitle: "SystemBrowser",

initialize: function($super) {
   $super();
   this.installFilter(new lively.ide.NodeTypeFilter(lively.ide.ClassFragmentNode), 'Pane1');
},

rootNode: function() {
   ide.startSourceControl();
   if (!this._rootNode)
   this._rootNode = new ide.SourceControlNode(tools.SourceControl, this);
   // this._rootNode = new ide.EnvironmentNode(Global, this);
```

Figure 5: The SystemBrowser browsing its own implementation

examine parameters. The control-flow, however, cannot be continued or stepped through like in a debugger due to the missing reflective features of JavaScript.



```
InspectHelper.inspect---
WikiNavigator.login---
CodeMarkupParser.setModelValue(setValue,false)
ButtonMorph.setValue(false)
ButtonMorph.onMouseUp(#<Event:MouseUp@pt(736.0,70.0)>,1265:
MouseHandlerForDragging.handleMouseEvent(#<Event:MouseUp@
```

```
functionName: setValue
newValue: false
```

```
CodeMarkupParser.setModelValue (functionName, newValue)
{
  if (this.formalModel)
   {
    var impl = this.formalModel[functionName];
    return impl && impl.call(this.formalModel, newValue);
   }
  var plug = this.getModelPlug();
  if (plug == null || plug.model == null || functionName == null)
   return null;
  var func = plug.model[plug[functionName]];
```

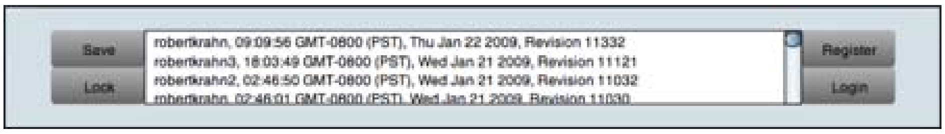
Figure 6: The StackViewer displays the method stack at user defined points

Additional Tools.

Lively Wiki provides a xUnit-like test framework for creating and running tests. An additional local code browser shows just the current changes of the world, so users don't have to deal with the rest of the code base. Additionally the support for profiling programs shows how often methods or functions where called and how much time was spent.

3.4 Wiki Control

To control the wiki-related properties users can open a tool called WikiNavigator by moving the mouse in the upper left corner of a world. With a WikiNavigator (see figure 7) users can register an account and login with their username, save and lock/unlock a world, and load versions of a world. The WikiNavigator also shows if a world is currently locked by another user.



```
robertkrahn, 09:09:56 GMT-0800 (PST), Thu Jan 22 2009, Revision 11332
robertkrahn3, 18:03:49 GMT-0800 (PST), Wed Jan 21 2009, Revision 11121
robertkrahn2, 02:46:50 GMT-0800 (PST), Wed Jan 21 2009, Revision 11032
robertkrahn, 02:46:01 GMT-0800 (PST), Wed Jan 21 2009, Revision 11030
```

Figure 7: The WikiNavigator

The explicit locking mechanism ensures that other users do not overwrite a world that is currently worked on. However, it is not mandatory to lock a world in order to edit it. This makes it possible that conflicts can appear: User A opens a world after user B has opened the same world. When A saves the world then B does not work on the most recent version of the world anymore. If B wants to save the page, all changes of A would be lost accidently. To prevent this scenario the WikiNavigator informs B that a more recent version of the world exists.

4. IMPLEMENTATION OF LIVELY WIKI

Lively Wiki extends the Lively Kernel in various ways. It implements a complete programming environment that allows to develop the wiki and the Lively Kernel from inside itself. Lively Wiki also interfaces WebDAV and Subversion to allow saving and versioning worlds. Using WebDAV to the full extent has also the advantage of being able to inspect and analyze wiki contents.

4.1 WebDAV and SVN Interface

The Lively Kernel is an almost completely client-based system. The user starts it by navigating to a world (currently a xhtml page) which will download and run the JavaScript source code and boot the system. From that moment all the application logic is gathered in the Web browser environment (of course requests to the server can be made using asynchronous JavaScript).

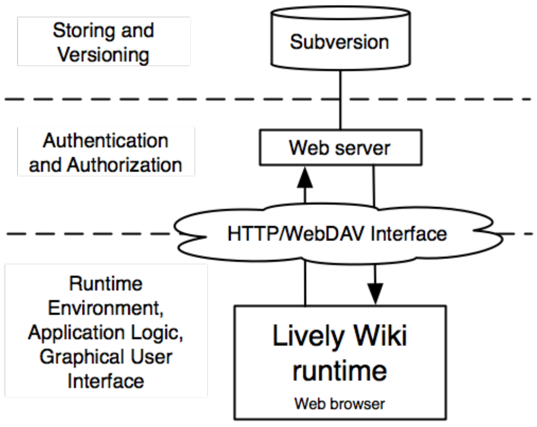

Storing and Versioning
Subversion
Authentication and Authorization
Web server
HTTP/WebDAV Interface
Runtime Environment, Application Logic, Graphical User Interface
Lively Wiki runtime
Web browser

Figure 8: Lively Wiki architecture

Figure 8 shows the system parts and their functions. We extended the system by connecting a Subversion repository on the server side in which worlds can be stored in a serialized form (Apache server version 2.2 in combination with the dav_svn module is used). We are using XHTML documents and append the world state as a SVG subtree. Using Subversion has the advantage that worlds are automatically versioned and meta data like author and and last modified date can be retrieved. To use those features from a Web browser we are using the WebDAV and DeltaV protocols. WebDAV is a extension of the HTTP protocol to support distributed authoring and versioning in the Web [1, 32] and DeltaV is an extension of WebDAV for advanced versioning features [37]. In detail, Lively Wiki uses the following HTTP/WebDAV/DeltaV methods:

- PUT. Upload the serialized contents to the server and create a new document or a new version of an existing document. It is used with the IF-Header to ensure that newer versions of a world are not overridden by accident.
- DELETE. For removing worlds from the repository.
- PROPFIND. Retrieve meta data from a version of a world, it is used to find out which is the most recent version of a world, who is the author, and what is the modification date. In combination with LOCK and UNLOCK it is also used to test if a world was locked.
- REPORT. Retrieve meta data from a collection of worlds. PROPFIND requests can only retrieve meta data from one resource and one version of that resource at a time. To find out about the version history of a world would require to issue several PROPFIND requests. The REPORT method allows to combine them into one request.
- LOCK and UNLOCK. Temporarily disable write access to a world to avoid editing conflicts. User can force the wiki to ignore locks.

To implement authorization and authentication Lively Wiki uses Apache's basic access control scheme. Currently normal wiki users can only modify worlds (XHTML documents), not JavaScript or other resources.

4.2 Tools

The development and wiki tools are all implemented using Morphic and Lively Kernel's observer model. The usual architecture for a tool is the following: A widget used to define the user interface and connect it with domain objects. Morphs are assembled in a buildView method and they are linked to the widget or to arbitrary domain objects. This allows to specify data flow conveniently without having to write unnecessary glue code and, at the same time, leaves domain objects independent from user interface definition.

The SystemBrowser uses OMeta [41] to parse JavaScript sources. This allows to organize definitions of JavaScript objects semantically, thereby allowing operations like searches for senders/implementors of methods or references of a class as well as simple refactorings such as the Move Method refactoring [9]. The SystemBrowser itself is implemented using a OmniBrowser-like architecture [5], this allows to keep the browser, its content, and supported operations extendable. Currently the browser is able to display source code of plain JavaScript files, Lively Kernels lkml definitions, and OMeta grammars.

5. APPLICATION EXAMPLES

To illustrate what can be done with Lively Wiki we will describe two applications build entirely in our system.

5.1 Calendar

As mentioned in the introduction a rapid application development platform would be convenient for several scenarios. For example it could be useful for a project to have a custom made calender application fulfilling certain requirements that are not provided by existing calendar applications. The finished application is shown in Figure 9: One wiki world represents one month and each day of the month is a rectangular shaped morph with a date attached to it. Users of the calendar can drag and drop notes into day morphs. The morph representing the current date is highlighted as well those day morphs that have notes attached to it and are in the period of the next seven days after the current date. In the following we present the steps for building this simple calendar in the wiki.

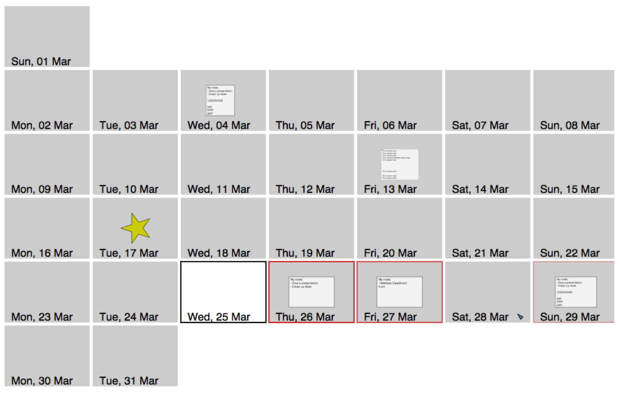


Figure 9: Wiki calendar application

To get started a new world has to be created. We do this by visiting an existing world (like the simpleObjects world[3]), open the world menu by right clicking into the background, and choosing 'publish as...'. In the new world we open a SystemBrowser for the local changes (the ChangeSet) of the world by open the World menu and choosing 'Tools...' and 'Local code browser'. Since the simpleObjects world had no changes attached the browser will show no items.

The calendar application has two types of objects: a morph representing a date and an object that creates the day morphs of month and arranges them properly on a screen. Using the context menu of the code browser we create a new class DayMorph inheriting BoxMorph. At this point it should be mentioned that changes have an immediate effect and can be tried out at once. For creating a DayMorph we evaluate `new DayMorph(new Rectangle(0,0,100,100)).openInWorld()` and a black rectangle appears in the upper left corner of the world. To customize the look and the behavior of the DayMorph we add a field named `style` to the morph. This defines declaratively the appearance of the morph and overwrite its constructor method so that a date object can be passed to a DayMorph upon creation:

```
style: {
   borderColor: Color.red,
   strokeOpacity: 0,
   fill: Color.lightGray
},
initialize: function($super, date, optPos) {
   $super(optPos.extent(this.initialExtent));
   var label = new TextMorph(this.bounds(),
      date.toString).beLabel();
   this.addMorphFront(label);
}
```

This source code can be written inside the bottom pane of the browser when the DayMorph class is selected. Accepting those changes with alt+s will save and evaluate them. In the same way we are overwriting the `addMorph` method to scale down text when it is dropped into a DayMorph:

```
addMorph: function($super, morph) {
   $super(morph);
   this.oldScale = morph.getScale();
   var scalePt = this.getExtent().scaleByPt(
      morph.getExtent().inverted());
   var scaleFactor = Math.min(scalePt.x,
        scalePt.y);
   if (scaleFactor < 1) morph.setScale(
        scaleFactor);
   morph.centerAt(this.getCenter());
   return morph;
},
```

Everything thats missing now is to overwrite the `onMouseDown` method to restore the scale (using the oldScale value) and a method that regularly checks the date of a morph against the current date and modifies the style. Morphs can register recurring actions in the system scheduler, this will call the specified method in intervals. Additionally a simple layouter is needed for creating DayMorphs with dates. For space reasons we won't present the rest of the source code here but refer the interested reader to `\url{http://livelykernel.sunlabs.com/repository/lively-wiki/livelyCalendar.}`
`xhtml`. Such a small but powerful application requires little code (the full example has around 70 LOC) and wiki programmers have to know nothing more than the Morphic interface.

5.2 Wiki Visualization

Having several entry points to worlds in a wiki can be useful [23]. We therefore created another application in the wiki that shows a live view of all wiki worlds, their links, and meta information. Figure 10 shows the graph: Each node represents a world, the size of the world indicates how much versions a world has and the color of a node is a indicator of how recent the last change was. Red nodes represent worlds modified just now, white nodes represent worlds that were not modified during the last 60 days.

The visualization uses a relaxation technique to arrange the nodes and is updated every few seconds to give a spectator not only a live but a lively view of the wiki. The visualization world can be found at `http://livelykernel.sunlabs.com/repository/lively-wiki/livelyWikiVis.xhtml`.

[3]`\url{http://livelykernel.sunlabs.com/repository/lively-wiki/simpleObjects.xhtml}`

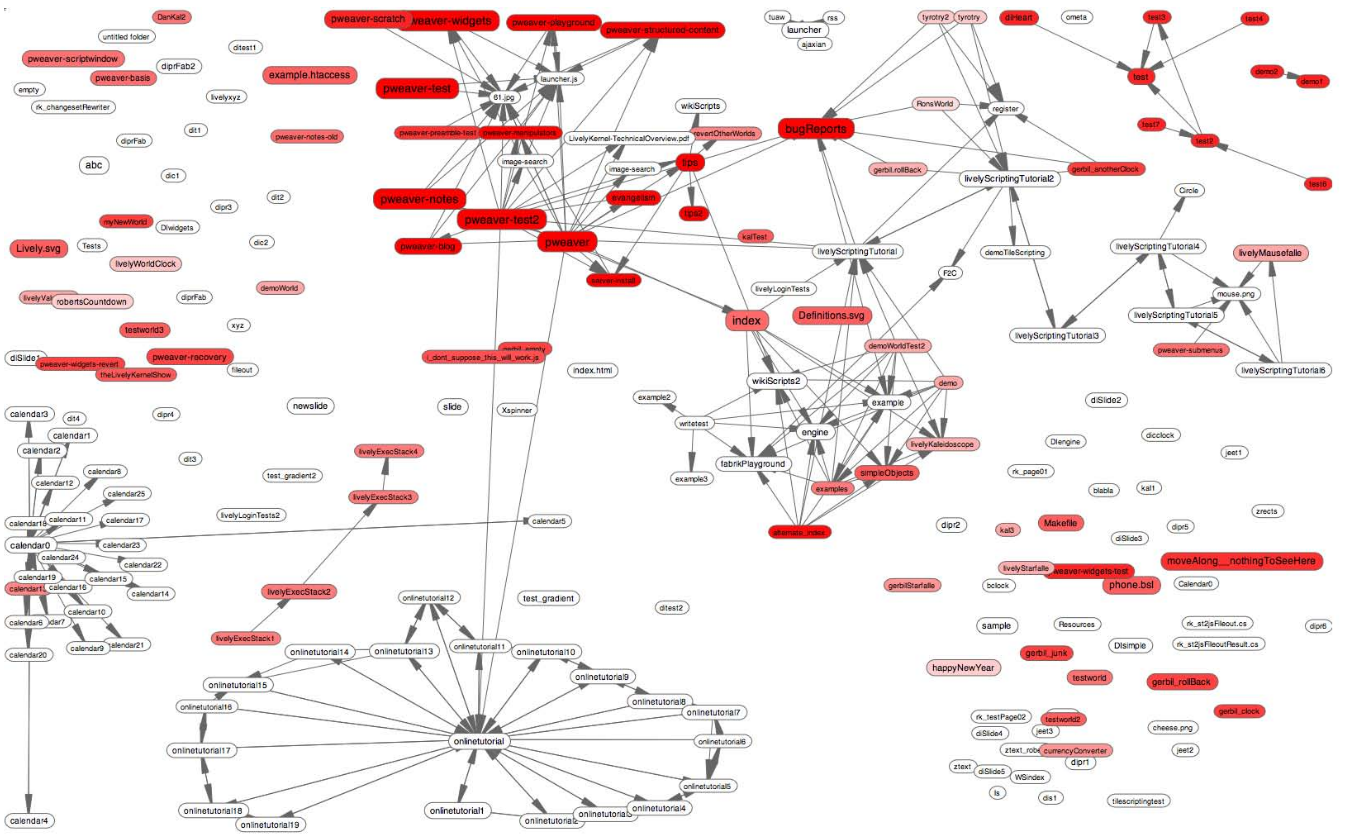


Figure 10: Wiki application that shows a live view of the existing wiki worlds, their links, and meta information

6. RELATED WORK

As described in section 2 Lively Wiki uses a client side Web development approach that is related to the following concepts and implementations.

6.1 Wikis

Wikis [23, 7] are one of the oldest of the Web application types that allow users to change Web content. Wikis are used for collaboration and information sharing. Wikis have an extensible set of pages whose content and organization can be changed by users [23]. Since their invention 1995 wikis have become popular [25, 40] and are used in corporate, academic, and personal contexts [38]. Usually the page content of a wiki page is hypertext with embedded pictures and attachements. The page can be edited with a markup language, but the page layout cannot be changed other than in a predefined manner. Typical wikis have no programming functionality to define dynamic behavior.

6.2 Application Wikis

Application Wikis [4, 33] extend normal wikis with lightweight programming features for the creation of ad-hoc dynamic content. As situational applications they solve an immediate, specific problem: for example they are used to integrate database access and business processes [33, 17, 2]. Their target group are domain experts who are often not professional programmers. As such they make programming tasks straightforward by embedding a scripting language to specify dynamic actions and behavior into the wiki markup language or providing visual programming capabilities [17]. This approach, however, is restricting because language constructs are often domain specific (like in [4]) and users are bound to the form of writing code in or inside markup languages without being able to interact and inspect objects. Lively Wiki gives users a simpler and more immediate feedback while programming and users can create general purpose applications.

6.3 Mashups

Mashups are applications that dynamically combine content from Web sites into something new [34]. There are a number of Web-based mashup creation tools. They allow end-users to program using textual- and visual languages, e.g. Yahoo Pipes [43], Microsoft Popfly [28], or Google Mashup Editor [16]. For a thorough review of Web-based mashup creation tools see [34]. All mashup development tools seem to be domain specific and do not allow to develop general purpose applications.

6.4 Active Web Essays

Active Web Essays such as Chalkboard [44] are based on the Active Essay concept [21]. They are educational tools where complex mathematical and scientific ideas are explained with modifiable dynamic simulations. Similar to Literate Programming [22] scripts define the behavior of the simulations, are at the same time part of the text, and can be changed by the user. Current Active Web Essays lack the directness of Lively Wiki and dedicated development tools.

7. SUMMARY AND OUTLOOK

We have designed and implemented Lively Wiki, a Web-based development platform supporting the collaborative creation of interactive and dynamic Web content. Instead of having to integrate lots of diverse technologies to create Web applications, Lively Wiki users can work with simple a interface in a live environment. Lively Wiki follows wiki principles to make application development as simple as possible.

We started to integrate visual data-flow programming tools [24] into the wiki for simplifying application development. We plan to extend those capabilities with other visual programming approaches like Tile Scripting [42] and alternative programming language that are easier to use than JavaScript.

Currently Lively Wiki supports only asynchronous collaboration. Implementing a synchronous collaboration model like Nebraska [18] or Kansas [29] would support interaction between developers. We also believe that collaboration could be greatly enhanced by providing automatic merging capabilities to worlds. When merging worlds, not only attached source code would be considered but also the scene graph so that morphs and the state of the world could be moved from one version to another.

Lively Wiki search capabilities are currently only implemented for the source code in a world (including the code base). As we found out in the wiki visualization experiment searching other worlds is straightforward and relatively fast by using XPath and Regular Expressions. We plan to extend the search function so that not only a normal text search in the whole wiki is possible but that also the local code of other worlds is searchable and can be browsed using the SystemBrowser.

8. ACKNOWLEDGEMENTS

We would like to thank Philip Weaver and Peter Fraser for their ongoing effort of testing Lively Wiki and for creating interesting applications. We would also like to express thanks for the constructive discussions which we had with Michael Perscheid, Michael Haupt, Malte Appeltauer, and Bastian Steinert.

9. REFERENCES

[1] RFC 4818 WebDAV Specification. `http://www.webdav.org/specs/rfc4918.html`, 2007. As of Feb 10 2009.

[2] XWiki. `http://www.xwiki.com/`, 2008. As of Mar 12 2009.

[3] Hypercard. `http://en.wikipedia.org/wiki/Hypercard`, January 2009. As of Jan 29 2009.

[4] Craig Anslow and Dirk Riehle. Towards End-User Programming with Wikis. In *WEUSE '08: Proceedings of the 4th international workshop on End-user software engineering*, pages 61–65, New York, NY, USA, 2008. ACM.

[5] Alexandre Bergel, Stephane Ducasse, Colin Putney, and Roel Wuyts. Meta-driven browsers. pages 134–156. Springer, 2007.

[6] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered web pages. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 163–172, New York, NY, USA, 2005. ACM.

[7] Ward Cunningham. WikiWikiWeb. `http://c2.com/cgi/wiki`, March 1995. As of Jan 28 2009.

[8] Ward Cunningham. Design Principles of Wiki: How can so little do so much?, 2006. Keynote at WikiSym 2006, `http://c2.com/doc/wikisym/WikiSym2006.pdf`.

[9] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman, Amsterdam, 1999.

[10] Martin Fowler. GUI-Architectures. `http://martinfowler.com/eaaDev/uiArchs.html`, 2008. As of May 31 2008.

[11] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[12] Athula Ginige and San Murugesan. Web engineering: an introduction. *Multimedia, IEEE*, 8(1):14–18, Jan-Mar 2001.

[13] Adele Goldberg. *SMALLTALK-80: the interactive programming environment*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[14] Adele Goldberg. Programmer as reader. *IEEE Softw.*, 4(5):62–70, 1987.

[15] D. Goodman. *The complete Hypercard handbook*. Bantam Books, Inc., New York, NY, USA, 1988.

[16] Google. Google Mashup Editor. `http://editor.googlemashups.com/`, 2008. As of Feb 10 2009.

[17] IBM. IBM Mashup Center. `http://www-01.ibm.com/software/info/mashup-center/`, 2008. As of Apr 02 2009.

[18] Dan Ingalls. Nebraska. `http://wiki.squeak.org/squeak/1356`. As of Mar 16 2009.

[19] Dan Ingalls, Ted Kaehler, John H. Maloney, Scott Wallace, and Alan Kay. Back to the Future: The Story of Squeak, a Practical Smalltalk Written in Itself. *ACM SIGPLAN Notices*, 32(10):318–326, 1997.

[20] Daniel Ingalls, Krzysztof Palacz, Stephen Uhler, Antero Taivalsaari, and Tommi Mikkonen. The Lively Kernel A Self-supporting System on a Web Page. In Robert Hirschfeld and Kim Rose, editors, *S3*, volume 5146 of *Lecture Notes in Computer Science*, pages 31–50. Springer, 2008.

[21] Alan Kay. Active essays. `http://web.archive.org/web/20060710213801/http://www.squeakland.org/whatis/a_essays.html`, 2006. As of Mar 3 2009.

[22] Donald E. Knuth. Literate programming. *Comput. J.*, 27(2):97–111, 1984.

[23] Bo Leuf and Ward Cunningham. *The Wiki way: quick collaboration on the Web*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[24] Jens Lincke, Robert Krahn, Dan Ingalls, and Robert Hirschfeld. Lively Fabrik - A Web-based End-user Programming Environment. In *In Proceedings of the*

Conference on Creating, Connecting and Collaborating through Computing (C5). IEEE, January 2009.

[25] Panagiotis Louridas. Using wikis in software development. *IEEE Software*, 23(2):88–91, 2006.

[26] John H. Maloney. *Morphic: The Self User Interface Framework*. Sun Microsystems, Inc., 1995.

[27] John H. Maloney and Randall B. Smith. Directness and Liveness in the Morphic User Interface Construction Environment. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 21–28, New York, NY, USA, 1995. ACM.

[28] Microsoft. Popfly. `http://www.popfly.com`, 2008. As of Sep 23 2008.

[29] Sun Microsystems. The Kansas Project. `http://research.sun.com/ics/kansas.html`. As of Mar 16 2009.

[30] Robert C. Miller. End User Programming for Web Users. *Workshop on End User Devlopment*, 2003.

[31] Yoshiki Ohshima, Takashi Yamamiya, Scott Wallace, and Andreas Raab. Tinlizzie wysiwiki and wikiphone: Alternative approaches to asynchronous and synchronous collaboration on the web. Technical report, Viewpoints Research Instititute, 2007.

[32] Mary Ellen O'Shields and Philip J. Lunsford II. WebDAV: A Web-Writing Protocol and More. *Journal of Industrial Technology*, 20(2).

[33] Dirk Riehle. End-User Programming with Application Wikis: A Panel with Ludovic Dubost, Stewart Nickolas, and Peter Thoeny. In *Proceedings of the 2008 International Symposium on Wikis (WikiSym '08)*. ACM Press, 2008. Pre-conference panel summary.

[34] Antero Taivalsaari. Mashware: The future of web applications. Technical report, Sun Microsystems, Feb 2009.

[35] Antero Taivalsaari, Tommi Mikkonen, Dan Ingalls, and Krzysztof Palacz. Web Browser as an Application Platform: The Lively Kernel Experience. Technical Report SMLI TR-2008-175, Sun Microsystems, January 2008.

[36] Mark J. Taylor, J. McWilliam, H. Forsyth, and S. Wade. Methodologies and website development: a survey of practice. *Information and Software Technology*, 44(6), April 2002.

[37] The Internet Society. RFC 3253 DeltaV Specification. `http://www.webdav.org/specs/rfc3253.html`, 2002. As of Feb 10 2009.

[38] K. T. L. Vaughan, Jon Jablonski, Cameron Marlow, Sunir Shah, and Ross Mayfield. Beyond the sandbox: Wikis and blogs that get work done. In *PROCEEDINGS OF THE ANNUAL MEETING-AMERICAN SOCIETY FOR INFORMATION SCIENCE*, volume 41, page 596. Information Today; 1998, 2004.

[39] Iwan Vosloo and Derrick G. Kourie. Server-Centric Web Frameworks: An Overview. *ACM Comput. Surv.*, 40(2):1–33, 2008.

[40] Christian Wagner. Wiki: A Technology for Conversational Knowledge Management and Group Collaboration. *Communications of the Association for Information Systems (Volume13, 2004)*, 13:265–289, 2004.

[41] Alessandro Warth and Ian Piumarta. OMeta: an Object-Oriented Language for Pattern Matching. In *DLS '07: Proceedings of the 2007 symposium on Dynamic languages*, pages 11–19, New York, NY, USA, 2007. ACM.

[42] Alessandro Warth, Takashi Yamamiya, Yoshiki Ohshima, and Scott Wallace. Toward A More Scalable End-User Scripting Language. In *Proceedings of the Conference on Creating, Connecting and Collaborating through Computing (C5)*, pages 172–178, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[43] Yahoo. Pipes. `http://pipes.yahoo.com/pipes/`, 2008. As of Sep 23 2008.

[44] Takashi Yamamiya, Alessandro Warth, and Ted Kaehler. Active Essays on the Web. In *Proceedings of the Conference on Creating, Connecting and Collaborating through Computing (C5)*. IEEE, January 2009.